ARMY RESEARCH LABORATORY

# Scalability Analysis of Linear Equation Solvers for Sparse Positive Definite Systems

by Robert L. Davis, Brian J. Henz, and Dale R. Shires

**ARL-TR-3076** **September 2003**

**Army Research Laboratory**
Aberdeen Proving Ground, MD 21005-5067

**ARL-TR-3076** **September 2003**

# Scalability Analysis of Linear Equation Solvers for Sparse Positive Definite Systems

**Robert L. Davis, Brian J. Henz, and Dale R. Shires**
**Computational and Information Sciences Directorate, ARL**

| **REPORT DOCUMENTATION PAGE** | *Form Approved OMB No. 0704-0188* |
|---|---|

| | | |
|---|---|---|
| **1. REPORT DATE** *(DD-MM-YYYY)*<br>September 2003 | **2. REPORT TYPE**<br>Final | **3. DATES COVERED (From - To)**<br>June 2002–August 2002 |

| | |
|---|---|
| **4. TITLE AND SUBTITLE**<br>Scalability Analysis of Linear Equation Solvers for Sparse Positive Definite Systems | **5a. CONTRACT NUMBER** |
| | **5b. GRANT NUMBER** |
| | **5c. PROGRAM ELEMENT NUMBER** |
| **6. AUTHOR(S)**<br>Robert L. Davis, Brian J. Henz, and Dale R. Shires | **5d. PROJECT NUMBER**<br>BCHA06-3U21CL |
| | **5e. TASK NUMBER** |
| | **5f. WORK UNIT NUMBER** |
| **7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**<br>U.S. Army Research Laboratory<br>ATTN: AMSRL-CI-HC<br>Aberdeen Proving Ground, MD 21005-5067 | **8. PERFORMING ORGANIZATION REPORT NUMBER**<br>ARL-TR-3076 |
| **9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)** | **10. SPONSOR/MONITOR'S ACRONYM(S)** |
| | **11. SPONSOR/MONITOR'S REPORT NUMBER(S)** |

**12. DISTRIBUTION/AVAILABILITY STATEMENT**
Approved for public release; distribution is unlimited.

**13. SUPPLEMENTARY NOTES**

**14. ABSTRACT**
The U.S. Army Research Laboratory (ARL) is currently developing a suite of parallel codes to model liquid composite molding (LCM) manufacturing processes. This software suite utilizes the finite element method in order to model the LCM process, thus requiring the solution of sparse linear equations. Code profiles have revealed that, similar to other scientific computing codes, the majority of the execution time is spent solving large systems of linear equations. Accordingly, it is desirable to use the most efficient solver package or combination of packages to quickly solve large sparse symmetric positive definite systems of equations as found in the LCM simulation software. A collection of linear equation solvers is being developed at ARL that the process simulation code accesses in order to automatically select the optimal solver for the given problem at runtime. The optimal solver is determined by considering factors such as architecture type, number of processors, matrix size and type, etc. This report evaluates several different linear equation solver packages to determine their applicability to this and other unstructured grid problems. Several factors, including accuracy, error, scalability, and runtime, are analyzed and reported.

**15. SUBJECT TERMS**
high performance computing, parallel processing, sparse linear equations solver, sparse symmetric matrices, resin transfer molding, finite element method

| **16. SECURITY CLASSIFICATION OF:** | | | **17. LIMITATION OF ABSTRACT** | **18. NUMBER OF PAGES** | **19a. NAME OF RESPONSIBLE PERSON**<br>Robert L. Davis |
|---|---|---|---|---|---|
| **a. REPORT**<br>UNCLASSIFIED | **b. ABSTRACT**<br>UNCLASSIFIED | **c. THIS PAGE**<br>UNCLASSIFIED | UL | 34 | **19b. TELEPHONE NUMBER** (*Include area code*)<br>(410) 278-5006 |

**Standard Form 298 (Rev. 8/98)**
Prescribed by ANSI Std. Z39.18

# Contents

## List of Figures

## List of Tables

## Acknowledgments

This research was made possible by a grant of computer time and resources by the Department of Defense High Performance Computing Modernization Program.

# 1. Introduction and Background

One common use of high performance computers today is to solve large complex problems in the physical sciences such as physics, chemistry, and engineering (*1*). The majority of these problems are formulated into mathematical models that represent the given physical situation. In many of these situations, an adequate model is obtained by using a finite number of well-defined components. These are "discrete" noncontinuous problems that can be solved even if the number of elements is very large (*2*). High performance computing can be used to obtain the solutions of these models faster and more efficiently than single processor workstations. The techniques used to get the solution of these problems are the basis of scientific computing.

Most mathematical models begin with some sort of physical problem. From this, certain factors, such as the conservation laws of physics, are considered (*1*). The construction of the model depends on retaining all factors that could affect the outcome, yet at the same time keeping the model as simple as possible (i.e., to keep everything that matters and nothing that doesn't). Once the model is constructed, it must be validated with both analytical solutions, if available, and experimental data.

Since analytical solutions are often not available, one must develop a numerical method for the solution (*1*). When developing a numerical method there are many factors to consider such as rounding errors, discretization error, and efficiency. Besides the numerical issues, there are software engineering issues that must be considered including the reliability, robustness, portability, and maintainability of the software used to implement the method. Once all of these issues are identified and understood the selection of the numerical packages, methods, and programming languages can proceed.

Parallel computing enhances scientific computing because of the potential increase in speed, size, and efficiency of problem solving (*3*). Multiple processors are analogous to a group of individuals assigned to work on a problem. Often, the group can work more efficiently than a single individual. Task partitioning and communication are used to assign a task to each processor and pass information from one processor to another.

However, even though we may find solutions to these problems through parallel programming, it alone is not sufficient (*4*). In order to solve larger and larger problems, the parallel algorithms must be scalable (the algorithms must be able to make efficient use of the additional resources). For example, a group may be working but not working well, one person in the group may be working less than the person next to him, just like some processors may be working less, thus slowing the group down as a whole. When work is evenly spread out, the group will perform at an optimal level thus getting the job done at a faster rate proportional to the added resources. When scalable algorithms are used, simulation times decrease enormously (days can shrink to hours). Many scientists use scalable algorithms to increase throughput. Scientific computing is largely focused on solving large linear systems of equations due to the large amounts of time devoted to this task in many codes.

This report details the results of an investigation into linear solver packages for solving the systems of equations typically found in the finite element method (FEM). Both iterative and direct solvers are analyzed. Iterative solvers refer to a wide range of successive approximations to obtain more accurate solutions to a linear system at each iteration. Packages such as the Portable, Extensible Toolkit for Scientific Computation (PETSc) include iterative methods including approaches such as the conjugate gradient (CG) and generalized minimum residual (GMRES) methods. Direct solvers obtain the exact solution in real arithmetic in a finite amount of operations. Packages such as the Parallel Sparse Symmetric Direct Solver (PSPASES) and the Vectorized Sparse Solver (VSS) use a direct method of solution, two examples of which are Gaussian elimination and Cholesky factorization. The various iterative and direct solvers are studied for comparison of performance data and profiling information such as solver time, iteration counts, error, floating point operations per second communication load, load balance, and scalability.

## 2. Solvers Investigated

In this section, a description of the various solver packages is investigated and presented. These solvers were chosen for availability of source code, common use, and free accessibility.

### 2.1 PETSc

PETSc is a combination of data structures and routines that provide the necessary basics for the implementation of very large application codes on both parallel and serial computers (*5*). PETSc includes parallel linear and nonlinear equation solvers, and quite a few of the mechanisms required in parallel application codes like parallel matrix and vector assembly routines. PETSc utilizes the message-passing interface (MPI) for all message-passing communication. It also interfaces with routines from various other packages for preconditioning and equation solving, although the packages are optional and do not need to be installed to use PETSc.

### 2.2 PSPASES

PSPASES is a parallel self-sufficient library based on MPI, made to solve a system of linear equations (*6*). It uses a direct method of solution consisting of four consecutive stages of processing: ordering, symbolic factorization, Cholesky factorization, and triangular systems solution. Using scalable and high performance algorithms, each of these phases are implemented. PSPASES must be used on parallel computers or networks of workstations (single processor implementation is not supported) that are equipped with MPI and basic linear algebra subprograms (BLAS) libraries and requires Fortran-90 and C language compilers.

### 2.3 VSS

VSS was originally developed to compute solutions for positive-definite systems in structural analysis software (*7*). More versions have since been developed for different applications aimed toward solving more general systems of equations. VSS uses a direct method with three steps to compute the problem solution: matrix reordering, matrix factoring, and forward/back solve.

### 2.4 COMPOSE Solver

ARL and its academic partners are currently developing a suite of codes to provide process modeling and simulation for the liquid composite molding (LCM) process. This suite, known as the Composite Manufacturing Process Simulation Environment (COMPOSE), is similar to other scientific codes in that the vast majority of the computation is performed in the solution of a large set of equations (*8*). COMPOSE uses the FEM and has been parallelized using the single program-multiple data methodology with MPI providing interprocessor communications (*9*). The problem domain is decomposed into non-overlapping sets of elements using a graph partitioner. An iterative preconditioned CG (PCG) algorithm, originally developed for use in multiple instruction-multiple data computers, is used (*10*). A Jacobi preconditioner is utilized in order to improve the convergence of the CG methods for the LCM problem (*11*).

## 3. Computing Environment

In this section, the computing environment that was used in the performance and scalability testing is discussed. This computing environment includes computer architecture, operating systems, compilers, and communication libraries.

### 3.1 Architecture

The current research and evaluations were conducted on an SGI Origin 3800[*] server system residing at the ARL Major Shared Resource Center. This distributed shared memory system uses a bristled hypercube interconnection network and scales to 512 processors. The current evaluations studied scalability up to 128 processors. The computer is populated with MIPS R12000 processors each clocked at 400 MHz. All simulation runs on this machine were made via a job submission and queuing system with the computer operating in production mode.

[*] SGI Origin 3800 is a registered trademark of Silicon Graphics, Inc.

### 3.2 Compiler Options

All executables were compiled with the MIPS compiler on the SGI system. Unless otherwise noted, all software was compiled with the `-O2` level of optimization that does extensive but conservative optimizations. This level of optimization enables certain global optimizations with the goal of increasing execution speed. Global optimizations typically include data flow analysis, strength reduction, and instruction scheduling optimizations.

For the analysis of the efficiency of the various approaches, different compiler optimizations are employed. The level of compiler optimizations was increased in some cases to `-O3` with some extending to the maximum of `-Ofast`. These levels of optimization result in aggressive optimization at the expense of increased code size. Code motion, loop unrolling, pipelining, and relaxed Institute of Electrical and Electronics Engineers arithmetic conformance are usually implied at these levels. The `-Ofast` level instructs the compiler to perform additional optimizations beyond the `-O3` level. These include interprocedural analysis and arithmetic reorderings. For example, the SGI compiler will allow for division splitting of $\frac{j}{k}$ into $j \times reciprocal(k)$ at this level, but not at the `-O2` or `-O3` levels.

### 3.3 Data Optimizations

For the parallel formulations, contiguous rows of the matrices are assigned to the same processor in block distribution format. With this in mind, several data optimizations are also performed on the matrices. These optimizations usually involve changes to the finite element mesh numbering scheme to allow for better hierarchical memory system performance. These changes do not impact the underlying simulation or solution of the linear systems of equations. These optimizations are described as they are presented.

As with many scientific codes running on hierarchical memory platforms, superlinear speedup can be observed in several of the test cases. Most often, this phenomenon is attributed to improved cache behavior as more processors are added to the pool. The resultant smaller memory requirements for each processor results in better memory locality and referencing. Because of this, several metrics presented are based on relative speedups and efficiencies to allow for more accurate comparisons.

## 4. Test and Evaluation

### 4.1 Small Matrix

First, a relatively small COMPOSE model with 105722 rows and 744590 non-zero elements was considered. The matrix is formed “as-is” from a finite element mesh package.

### 4.1.1 Unoptimized

No optimizations are performed on either the finite element mesh or the sparse matrix. This served as the initial test case for all the various packages. PETSc was the first package investigated, which includes 11 different Krylov methods with at least 11 preconditioners to choose from. Krylov methods are a class of iterative methods used to find the solution of sparse linear systems of equations. Preconditioners are used to improve the convergence of iterative methods. The 11 solution methods utilized are (1) CG, (2) conjugate gradient squared (CGS), (3) biconjugate gradient (BICG), (4) biconjugate gradient stabilized (BCGS), (5) Richardson, (6) Chebychev, (7) general minimized residual (GMRES), (8) conjugate residual (CR), (9) transpose-free quasi minimal residual–1, (10) transpose-free quasi minimal residual–2, and (11) the least-square method. All of these methods were run three times using a Jacobi preconditioner on 1, 2, 4, 8, 16, 32, and 64 processors. The same was done for PSPASES (from here on referred to as PSP in all figures) with the exception of the three jobs at one processor because PSP can only be used in parallel and with powers-of-two processors (i.e., 2, 4, 8, etc.). From all of the processor groups, the best time out of the three jobs was recorded.

When the results were collected, only 4 of the 11 PETSc methods converged to a solution. The four methods were CG, BICG, BCGS, and CR. CG generates a sequence of orthogonal or conjugate vectors representing the residuals of the iterates. These vectors are also the gradients of a quadratic function that, when minimized, is the same as solving the linear system (*12*). BICG generates two CG-type sequences of vectors: one based on $A$ (the original coefficient matrix) and the other based on $A^T$ (the transpose of $A$). By doing this they are mutually orthogonal and can be used to solve unsymmetric systems of equations. BCGS is a variant of biconjugate gradient, but obtains smoother convergence by using different updates for the $A^T$-sequence. The Jacobi preconditioner is found to be the best preconditioner for this problem. It consists of the square root of the diagonals of the stiffness matrix, and it is possible to use this preconditioner without using any storage other than that which the matrix provides. Out of the four PETSc methods that converged, the two that stood out of the group were CG and CR. They were the fastest (judged by wall clock time) and had the lowest iteration counts. Between the two, CR completed faster than CG and required fewer iterations. However, CG had the better error norm between the two. The other two approaches (BICG and BCGS) had much higher wall clock times compared to CR and CG (in many instances twice as high) and higher iteration counts as well. It should be noted though that BICG had the second best prior norm out of the four PETSc methods. Part of the reason for the poorer performance from BCGS and BICG is that CG and CR are designed specifically for symmetric matrix systems of equations. All test cases successfully completed, with the exception of the 4-processor BCGS run that halted prematurely with an error message.

PSPASES, being a direct solver, generates accurate solutions and does not require iterations as found in iterative solvers. Compared to PSPASES, none of the iterative methods could compete as they took much longer to solve. PSPASES at all numbers of processors was at least 11.46

times faster than CR. The top three methods (PSPASES, CR, and CG) all had relatively good scalability (between one and eight processors), but as the number of processors increased over eight the times did not scale as well. In fact, for all of the methods the times increased at some point between 16 and 64 processors. PSPASES outperformed all other approaches. Both its time and error norm, as seen in Figure 1, were better than any of the PETSc method results.

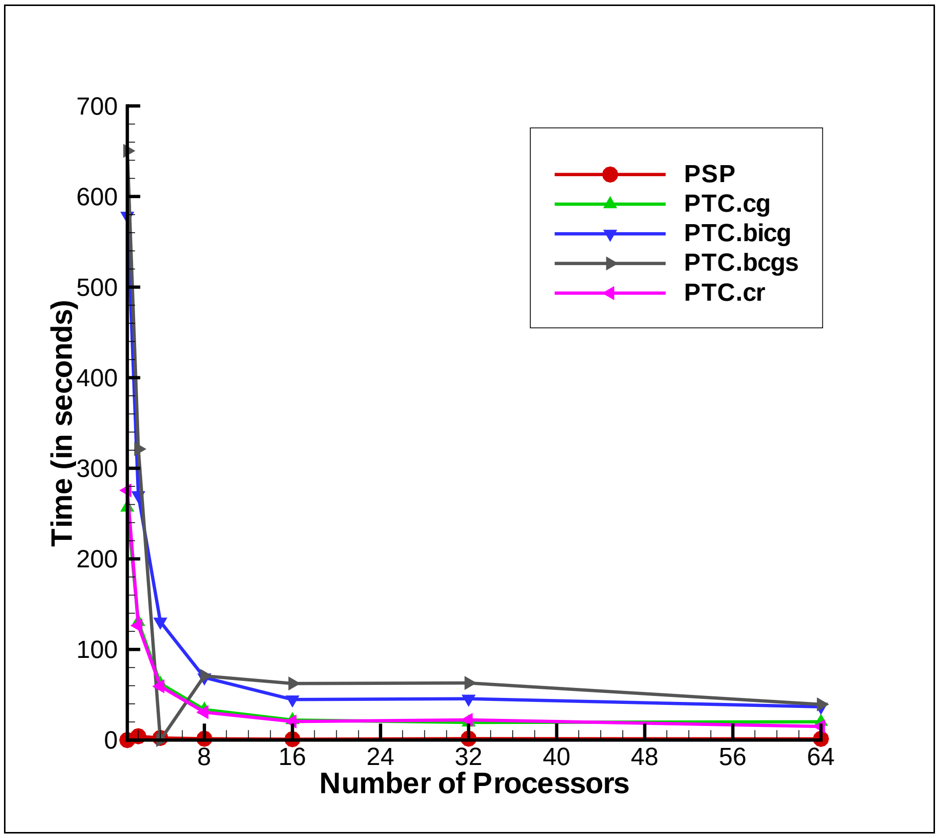

Figure 1. Small unoptimized matrix results.

### 4.1.2 Optimized

For the next set of tests, the same small matrix is used but it is optimized using a technique based on the Reverse Cuthill McKee approach. This method is typically used to minimize the bandwidth of the matrix (*8*). Reducing the bandwidth of the matrix often leads to better use of system cache. This technique of optimizing will also reduce communication load because of the partitioning method used (contiguous rows are assigned to individual processors). This, in turn, leads to fewer messages being sent and received and should improve scalability. The reason why there should be less communication is that more of the non-zero elements are in close proximity to each other and hence reside on the same processor.

Again the PETSc methods were the first to be tested, but this time only the methods that succeeded previously (CG, CR, BCGS, and BICG) were tested along with the direct solver, PSPASES. As predicted, by optimizing the matrix the solver times were significantly reduced over those of the unoptimized matrix results (Table 1). When analyzing the results from 2 to 64 processors, it was found that at one point, from the BICG method, the time at 32 processors was ≈74.6% faster than the time of the unoptimized matrix. These kinds of results were similar throughout the PETSc methods. CG had an average decrease of ≈35.6% with its best improvement at 64 processors, which was ≈57% faster than that of the unoptimized matrix as seen in Figure 2. CR had an average decrease of ≈36.6% with its best improvement at 32 processors that was ≈61% faster than that of the unoptimized matrix. BICG had an average decrease of ≈40.9% from the times of the unoptimized matrix. BCGS had the largest average decrease, which was ≈52.7% with its best improvement at 32 processors that was ≈65.9% faster than the unoptimized matrix results.

Table 1. Wall clock improvements through optimization.

| | **Percentage Increased Performance (Optimized vs. Unoptimized Matrix)** | | |
|---|---|---|---|
| | | **PETSc** | |
| **Number of Processors** | **PSPASES** | **CG** | **CR** |
| 2 | +8.4 | +21 | +16.4 |
| 4 | +2.9 | +25.1 | +24.7 |
| 8 | +12.1 | +26.7 | +25.2 |
| 16 | 0 | +35.7 | +35.7 |
| 32 | +44.4 | +47.8 | +61 |
| 64 | +8.3 | +57 | +56.5 |

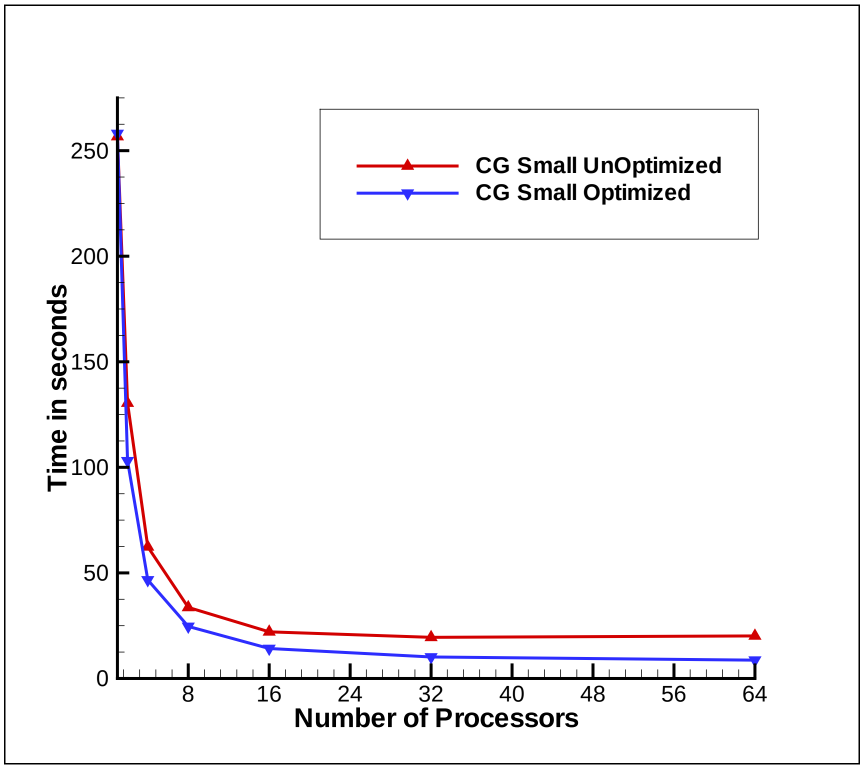

Figure 2. CG: optimized vs. unoptimized for the small matrix.

PSPASES improved with optimization, but the difference was minimal when compared with the PETSc methods. PSPASES had an average improvement of ≈15.2%, not including the time at 16 processors where it did not improve at all, and its best improvement was at 32 processors where it was ≈44.4% faster than that of the unoptimized matrix results (Figure 3). By optimizing the matrix, the times improved by ≈36.2% along with an improvement in scalability.

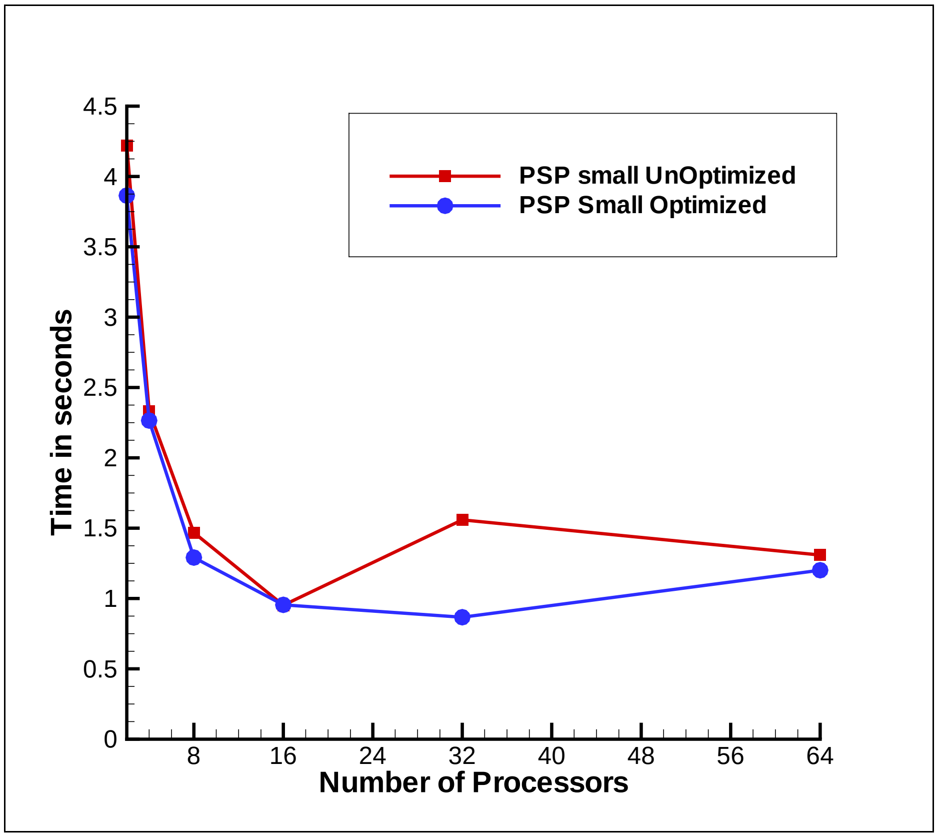

Figure 3. PSPASES: optimized vs. unoptimized for the small matrix.

There was a large decrease in the number of messages as well. On the average, between 2 and 64 processors there was a decrease of ≈78.7% between the optimized and unoptimized matrix with regard to the number of messages required. The best improvement was observed at 64 processors where the messages were ≈92.1% lower than that of the unoptimized matrix message results (Figure 4).

When choosing the best package from the optimized matrix, the results are similar to that of the unoptimized matrix. PSPASES was still the fastest solver package, but this time COMPOSE took second place edging out both CG and CR. However, the difference was that PETSc CG and CR were much closer to PSPASES than they were in the unoptimized matrix results (Figure 5).

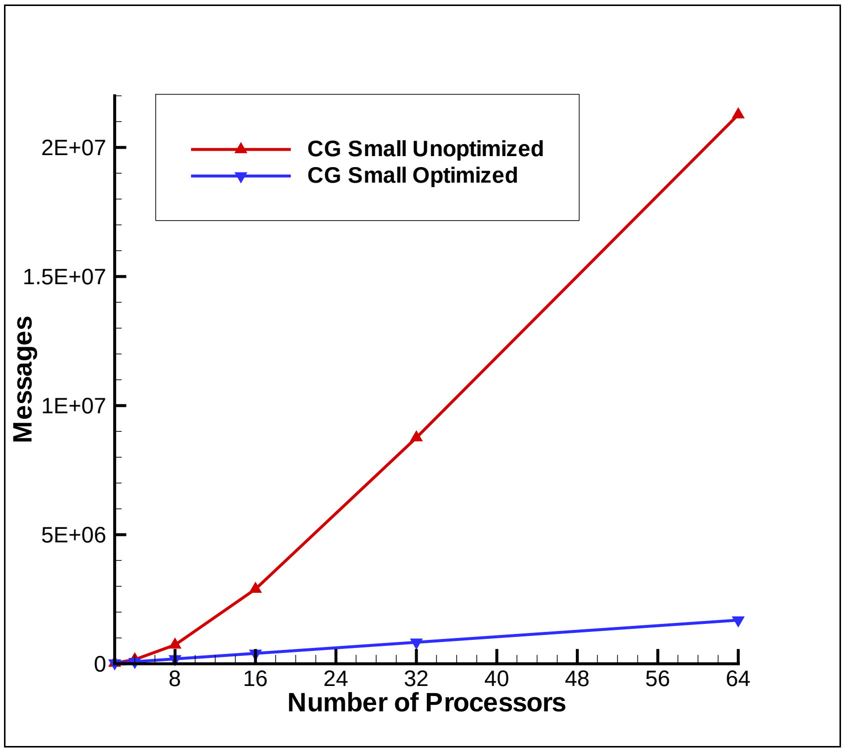

Figure 4. Messages: optimized vs. unoptimized for the small matrix.

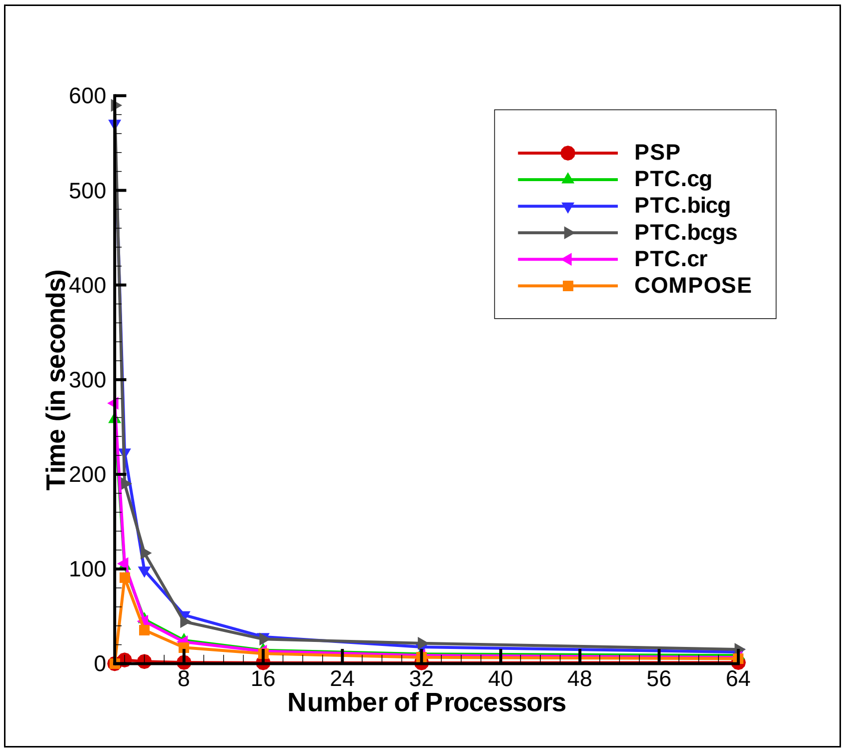

Figure 5. Small optimized matrix results.

The speedups achieved for the various packages are shown in Table 2. The formula for relative speedup is given as $\hat{S}_p = \frac{p_{\min} T_{p_{\min}}}{T_p}$, where $p_{\min}$ is the smallest number of processors that were used to compute a solution and $T_p$ is the time associated with the given processor count. This table highlights the fact that sometimes the more scalable approaches do not always represent the best solution. PSPASES scales poorly when compared to PETSc or COMPOSE; however, it requires significantly less wall clock execution time to arrive at the same solution. The relatively small size of this problem, combined with the properties of matrices stemming from sparse unstructured meshes, did not allow for a sufficiently sized work quantum at larger central processing unit (CPU) sets for effective parallelization. For the iterative methods used in COMPOSE and PETSc, the reduced work quantum at larger CPU sets, combined with extended interprocessor communications, results in a speedup curve that quickly levels with increased numbers of CPUs. While a detailed profile and analysis may provide more insight into the internal workings of these parallel techniques, it is clear that making the correct choice is always problem-dependent and usually involves considerations including the maximum number of CPUs available and exactness of the solution required.

Table 2. Scalability of various approaches.

| No. of Processors | PSPASES | | PETSc | | | | COMPOSE | |
|---|---|---|---|---|---|---|---|---|
| | | | CG | | CR | | | |
| | Time (s) | Speedup | Time (s) | Speedup | Time (s) | Speedup | Time (s) | Speedup |
| 8 | 1.291 | 8.0 | 21.938 | 8 | 20.411 | 8 | 17.042 | 8 |
| 16 | 0.955 | 10.88 | 12.614 | 13.92 | 11.84 | 13.76 | 10.667 | 12.8 |
| 32 | 0.867 | 11.84 | 8.727 | 20.16 | 8.0515 | 20.16 | 6.687 | 20.48 |
| 64 | 1.201 | 8.32 | 7.0395 | 24.96 | 6.154 | 26.24 | 5.295 | 25.6 |

### 4.1.3 Optimized With ATLAS Routines

The same optimized matrix was used for these tests, but this time automatically tuned linear algebra software (ATLAS) was used to tune the BLAS and LAPACK routines. ATLAS applies empirical techniques to the compilation of BLAS and LAPACK routines in order to provide portable performance. ATLAS optimizes BLAS and LAPACK routines for available cache. When using ATLAS, the BCGS solver in PETSc failed at 1, 2, and 32 processors returning an error message stating "Breakdown in Krylov Method!" This also happened once in the original matrix group at four processors. There were mixed results when the ATLAS software package was used. CG generally had faster times except at one processor (Figure 6). BICG had better times at 4, 8, and 16 processors, but slower times at 1, 2, 32, and 64 processors. BCGS was only faster at four processors and the rest of the processors times were either slower or failed to complete, and CR had faster times at 2, 4, 8, and 16 processors and had slower times at 1, 32,

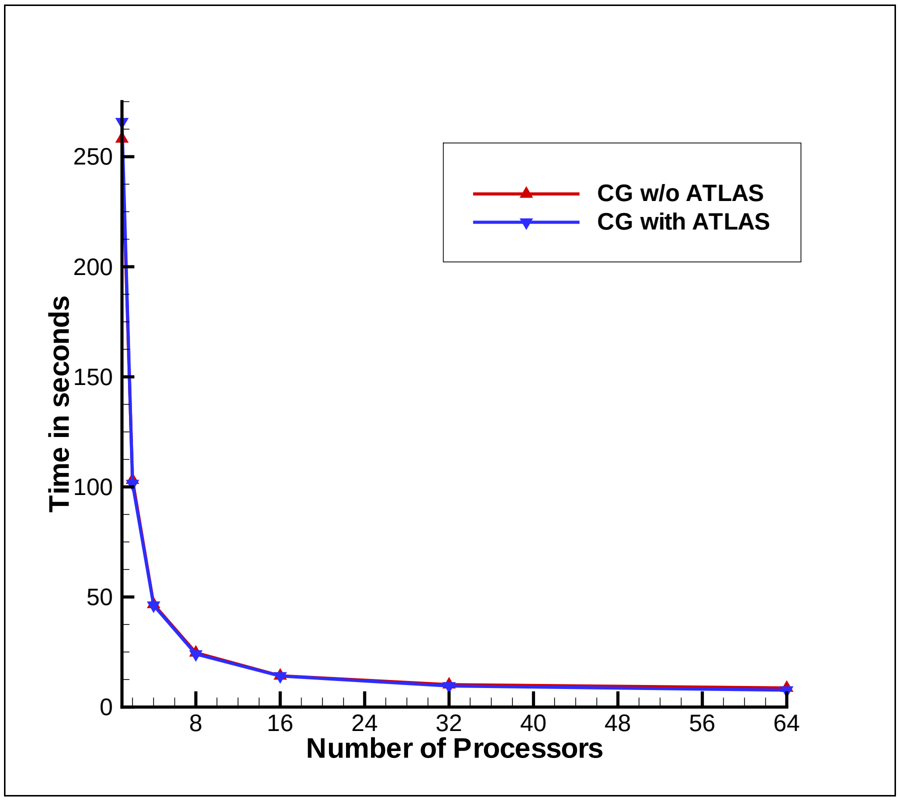

Figure 6.  CG:  ATLAS vs. non-ATLAS for the small matrix.

and 64 processors.  PSPASES did not improve at all.  It actually performed worse when using the ATLAS software package.

When observing the results between 2 and 64 processors, one notices that the performance change with ATLAS was minimal (Table 3).  For the BICG jobs that showed improved performance, there was about a ≈1.3% average improvement, and in the jobs where it did not improve, there was ≈9.3% average increase in wall clock time.  The most significant improvement among the jobs of BICG was at eight processors, where it improved by ≈2.3% and at 64 processors where it had the largest increase in time, which was ≈23% (these results are not shown in Table 3).  For CR in the jobs that improved, there was ≈2% average improvement, and

Table 3.  Effects of using ATLAS routines.

| | **Percentage Wall Clock Change (Optimized Matrix)** | | |
|---|---|---|---|
| | | **PETSc** | |
| **No. of Processors** | **PSPASES** | **CG** | **CR** |
| 2 | –3.6 | +1.6 | +1.4 |
| 4 | –2.4 | +1 | +2.6 |
| 8 | –8.1 | +2.4 | +2.4 |
| 16 | –14.8 | +0.7 | +1.6 |
| 32 | –9.5 | +5.9 | –3.5 |
| 64 | –1.5 | +11.5 | –10.1 |

for the jobs that did not improve there was ≈6.8% average increase in time. CR had its largest improvement at four processors where the time improved by ≈2.6%, and its largest increase of time at 64 processors where the time increased by ≈10.1%. PETSc CG and PSPASES were the two most affected by using the ATLAS software package. CG improved its solver time with an average of ≈3.9% over the regular optimized matrix results when using the ATLAS software package and had its most significant effect at 64 processors where the time improved by ≈11.5% (Figure 6). PSPASES did not improve its solver time and in fact showed an average increase in time of ≈6.6% over the regular optimized matrix results, and had its largest increase in time at 16 processors, which was ≈14.8% (Figure 7). All in all, PSPASES remained the optimal package choice (Figure 8).

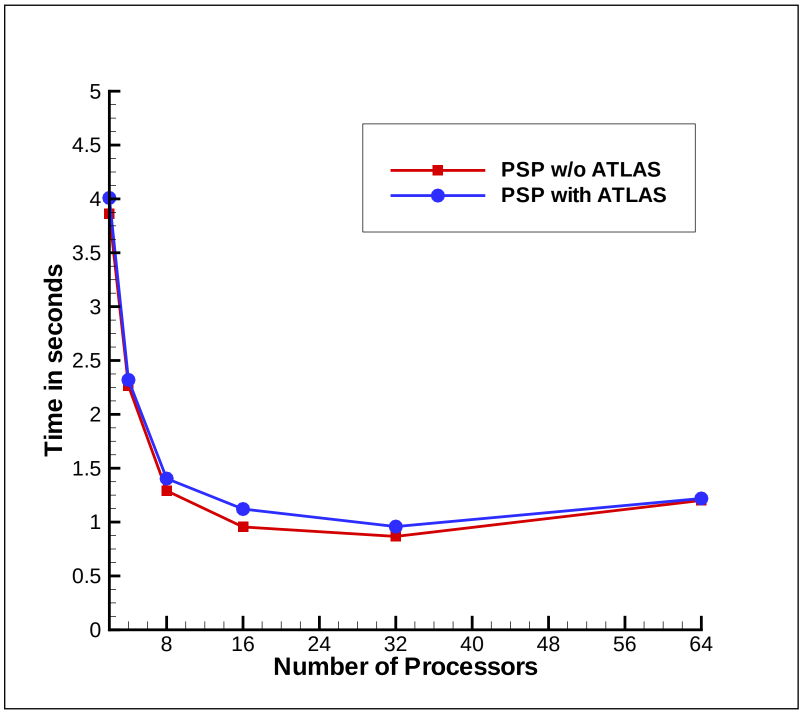

Figure 7. PSPASES: ATLAS vs. non-ATLAS for the small matrix.

### 4.1.4 Compose

The COMPOSE simulation software uses a preconditioned CG solver. Unlike the package solvers employed, COMPOSE uses the Metis graph partitioner (*13*) for domain decomposition with element-based rather than node-based partitioning. This makes a direct comparison with the package solvers difficult because they use node-based partitioning for domain decomposition. One of the difficulties of comparing the results includes the computation of the residuals in the iterative solvers. In the package solvers such as PETSc, the $L_2$ norm of the residual is computed over the entire domain rather than for each subdomain as in COMPOSE.

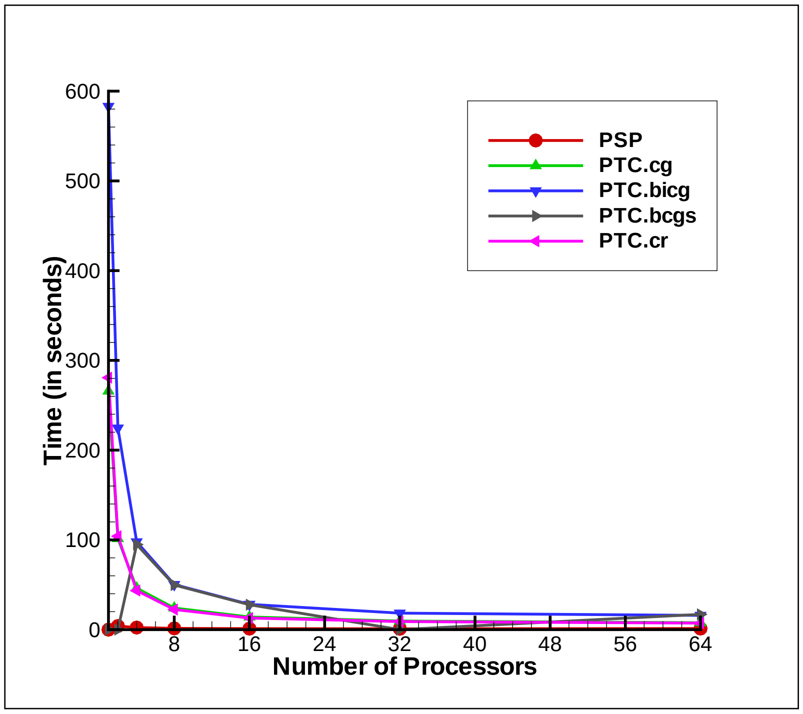

Figure 8. Small optimized ATLAS matrix results.

### 4.1.5 Summary

Figure 9 shows the timing data for the various packages using different compiler options and data manipulations. In Figure 6, PSP refers to timing results for the PSPASES solver, and PTC refers to timing results obtained from the PETSc solvers. CG and CR refer to the PETSc conjugant gradient and conjugate residual solvers, respectively. `-O3` and `-Ofast` refer to the compiler optimization levels used. The Metis options here refer to distributing the data based on a nodal partition of the graph formed from the finite element mesh. The primary idea is to achieve a better load balance and limit communication (the goals of the graph partitioner) as compared to the default block distributions.

We are satisfied with the results that we have received from testing these various packages on the small unoptimized, optimized, and optimized ATLAS matrices. We have determined that by partitioning a matrix with the Metis graph partitioner before testing it should improve the performance of the package solvers, particularly their scalability. We found that when using the ATLAS software package our results became inconsistent. Maybe if one wants to use the CG method for PETSc, they may want to use the ATLAS software package; otherwise, it didn't help for this problem. VSS is the linear solver used by the serial version of COMPOSE, which we tested with the optimized and unoptimized matrices. It outperforms all other methods except for PSPASES at one processor.

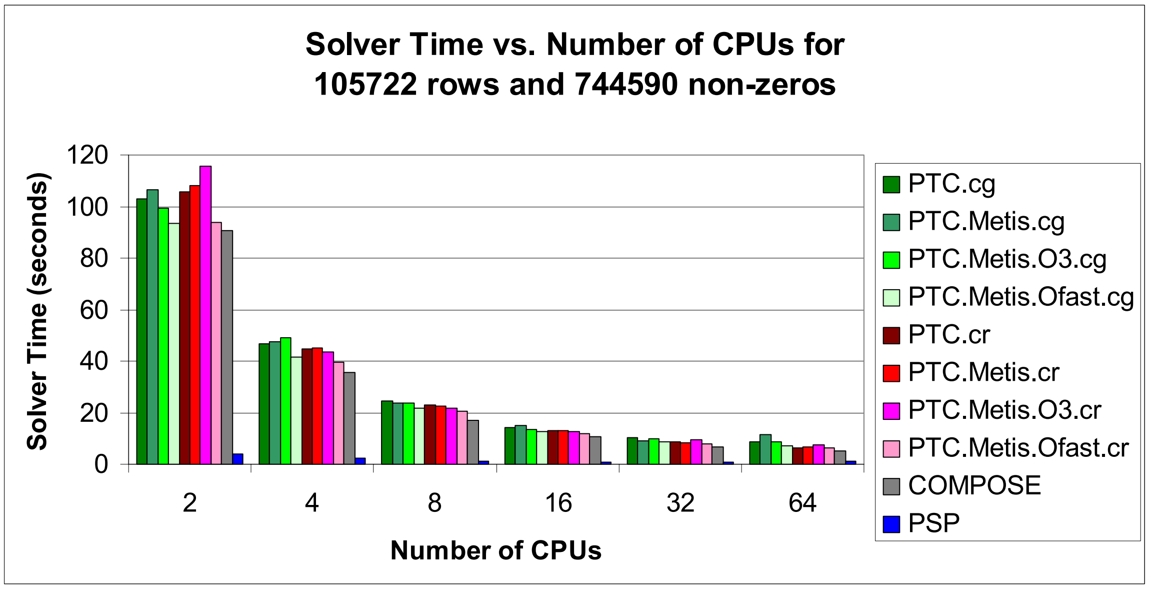

Figure 9. Small optimized matrix results including those with complier options.

Overall, we have found that when solving a relatively small linear system of equations, PSPASES is the best method for the application. ARL COMPOSE works very well, and PETSc CR and CG will work as well. But, when looking at the efficiency of each of the packages and solvers, as shown in Table 4, one can see that from a baseline of eight processors PSPASES has the worst relative efficiency at 64 processors. Relative efficiency is given as $\hat{E}_p = \dfrac{p_{\min} T_{p_{\min}}}{p T_p}$.

On the other hand, the PETSc methods and COMPOSE have an efficiency of more than double that of PSPASES. These numbers are critical in making the correct choices for processor pool sizes.

Table 4. Small matrix solution method efficiencies.

| No. of Processors | PSPASES | PETSc | | COMPOSE |
|---|---|---|---|---|
| | | CG | CR | |
| 8 | 1.0 | 1.0 | 1.0 | 1.0 |
| 16 | 0.68 | 0.87 | 0.86 | 0.80 |
| 32 | 0.37 | 0.63 | 0.63 | 0.64 |
| 64 | 0.13 | 0.39 | 0.41 | 0.40 |

## 4.2 Medium Matrix

A larger matrix containing 425,156 rows and 2,985,044 non-zero elements is used in this section. As before, the jobs are executed on the SGI Origin 3800 with similar data collection. Due to the larger problem size, the tests were scaled to 128 processors.

### 4.2.1 Unoptimized

The resulting times for the PETSc methods were as expected. For CG the time at 1 processor was ≈10.7 times longer, but at 64 processors the time was only ≈5.3 times longer than the times for the smaller matrix; for BICG the time at 1 processor was ≈10.3 times longer, but at 64 processors it was only ≈4.8 times longer. BCGS had a time that was ≈10.6 times longer at 1 processor and a time that was ≈5.4 times longer at 64 processors; CR had a time that was ≈9.7 times longer at 1 processor and a time that was ≈6.3 times as long at 64 processors. Overall, the PETSc method's times at 1 processor were ≈10.3 times longer and ≈5.5 times longer at 64 processors. For PSPASES, the times were even closer, with the time at 2 processors ≈4.9 times longer and 3.1 times longer at 64 processors. Also note that there are issues with cache at eight processors and lower.

When these numbers drop from ≈10 to ≈5 and from ≈5 to ≈3, it shows that when using a larger matrix the packages scale better. Again, even for the larger matrix, PSPASES is still the fastest between all of the solver packages. One thing to consider is the scalability between the PETSc methods and PSPASES. CR and CG times were still scaling as the number of processors grew all the way until 128 processors, while PSPASES times started to increase after its lowest time at 32 processors. What if we did not stop at 128; what would happen if we kept going up to 256, 512, or even 1024 processors? Maybe PETSc could outperform PSPASES. Up to 128 processors, PSPASES has the competition beat (Figure 10).

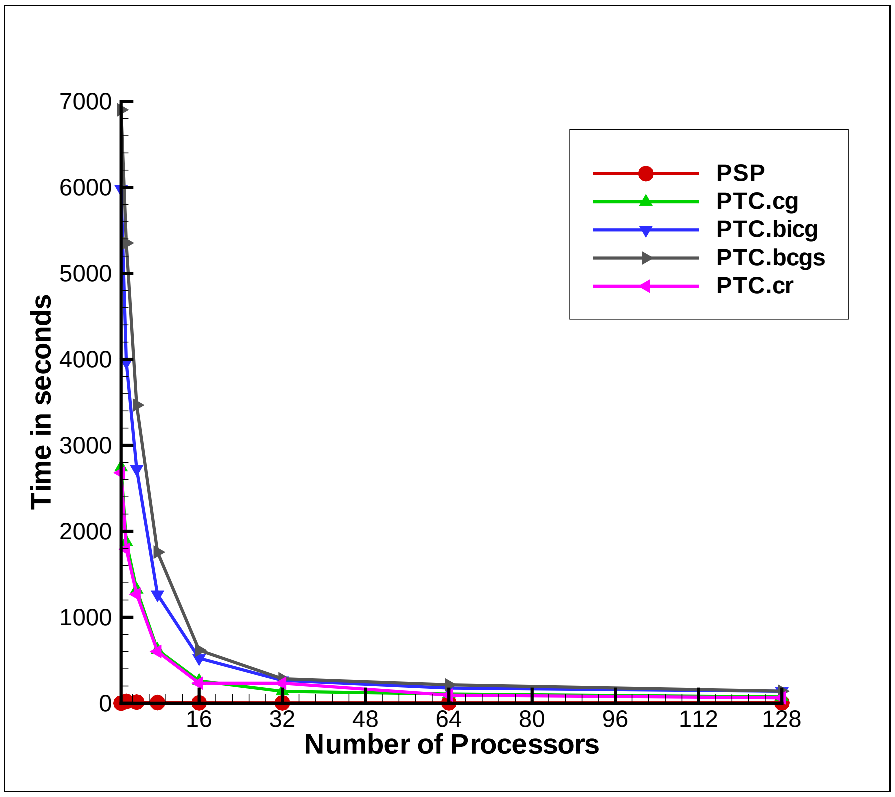

Figure 10. Medium unoptimized matrix results.

### 4.2.2 Optimized

After optimizing the medium matrix and submitting all of the jobs, it was expected to see the same kind of results as in the small optimized matrix: faster times and fewer messages. When the results came back, BCGS at 2, 4, 16, 64, and 128 processors gave the error message, "Breakdown in Krylov Method." This time BCGS results were not analyzed. The rest of the jobs ran well with noticeable differences in time and scalability between the optimized matrix results and the unoptimized matrix results (Table 5).

Table 5. Wall clock improvements through optimization.

| No. of Processors | Percentage Increased Performance (Optimized vs. Unoptimized Matrix) | | |
|---|---|---|---|
| | | PETSc | |
| | PSPASES | CG | CR |
| 4 | +17.4 | +49 | +45.6 |
| 8 | +16.3 | +62.8 | +61.1 |
| 16 | 0 | +59.8 | +58.4 |
| 32 | +16.8 | +58.2 | +56.8 |
| 64 | +25.7 | +65.8 | +58.4 |
| 128 | +20.8 | +55.7 | +51.1 |

When comparing the times from 4 to 128 processors, some improvement was observed when using the optimized matrix. CG had an average decrease in time of ≈58.6%, and its largest decrease at 64 processors, which was ≈65.8% between the optimized and unoptimized matrix results (Figure 11). BICG had an average decrease in time of ≈55.2%, with its largest decrease at 8 processors, which was ≈61.1%. CR had an average decrease in time of ≈59.6% and its largest decrease at 32 processors, which was ≈76.8%.

PSPASES again was not affected by the optimization as much as the PETSc methods were, though it did affect the medium matrix results more than the small matrix results. PSPASES had an average decrease in time of ≈19.4%, and its largest decrease at 64 processors was ≈25.7% (Figure 12).

Just like before, the number of messages decreased dramatically. On the average, there was a decrease in the number of messages of ≈78.7%, and the largest decrease came at 128 processors where the messages were down ≈93.6% over those of the unoptimized matrix (Figure 13).

As with the other results, PSPASES again was the fastest solver package with COMPOSE outperforming CG and CR of PETSc. The PETSc methods and COMPOSE all scaled better than PSPASES again. The PETSc and COMPOSE times monotonically decreased with the addition of processors up to 128 but the PSPASES time started to increase after 64 processors (Figure 14).

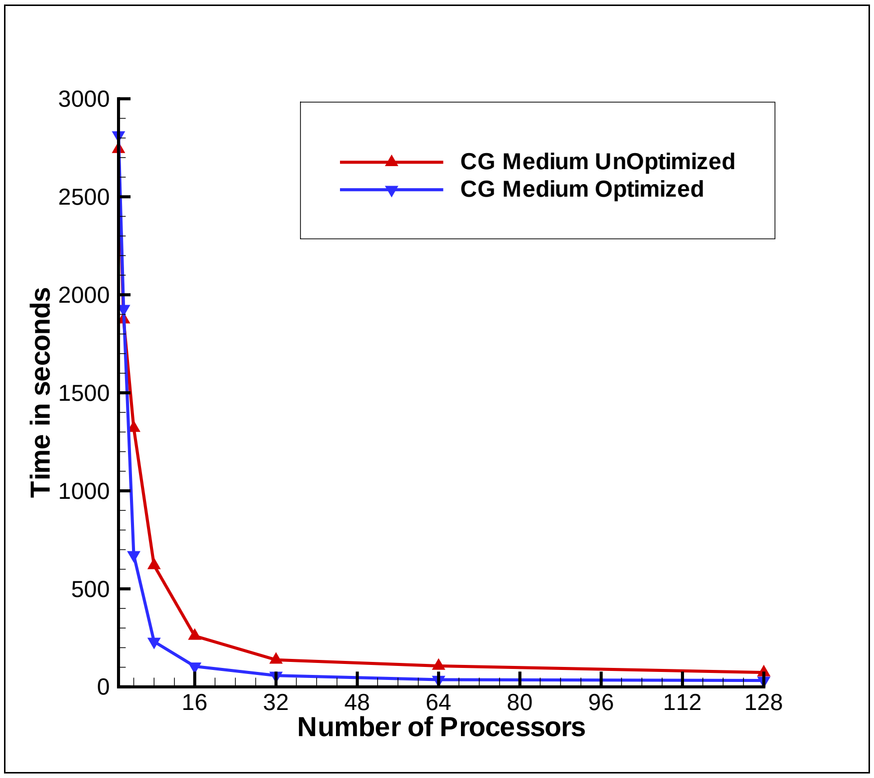

Figure 11. CG: optimized vs. unoptimized for the medium matrix.

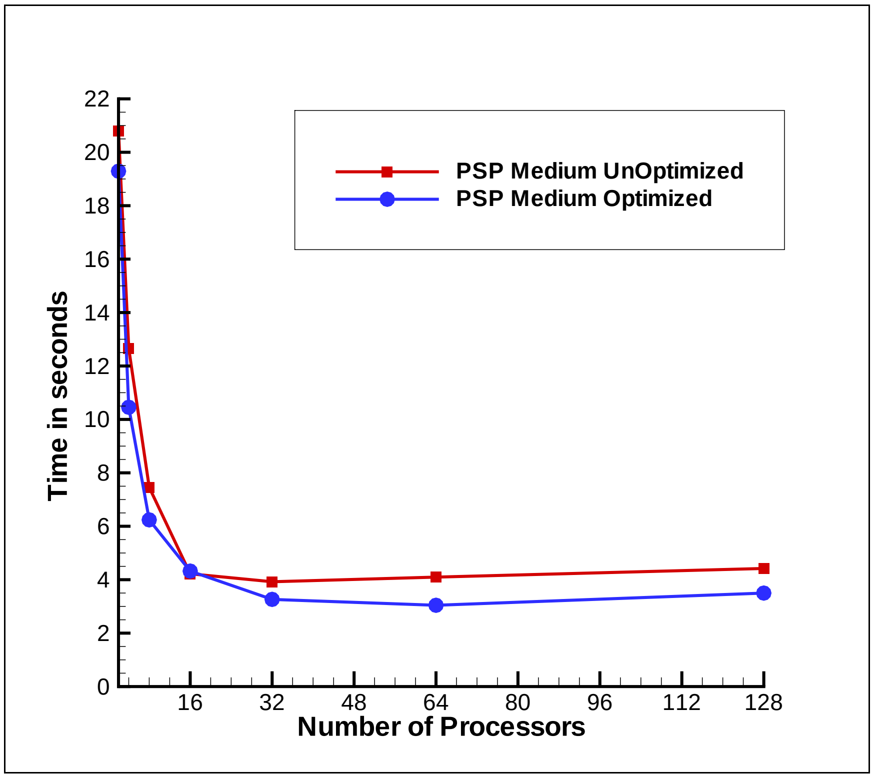

Figure 12. PSPASES: optimized vs. unoptimized for the medium matrix.

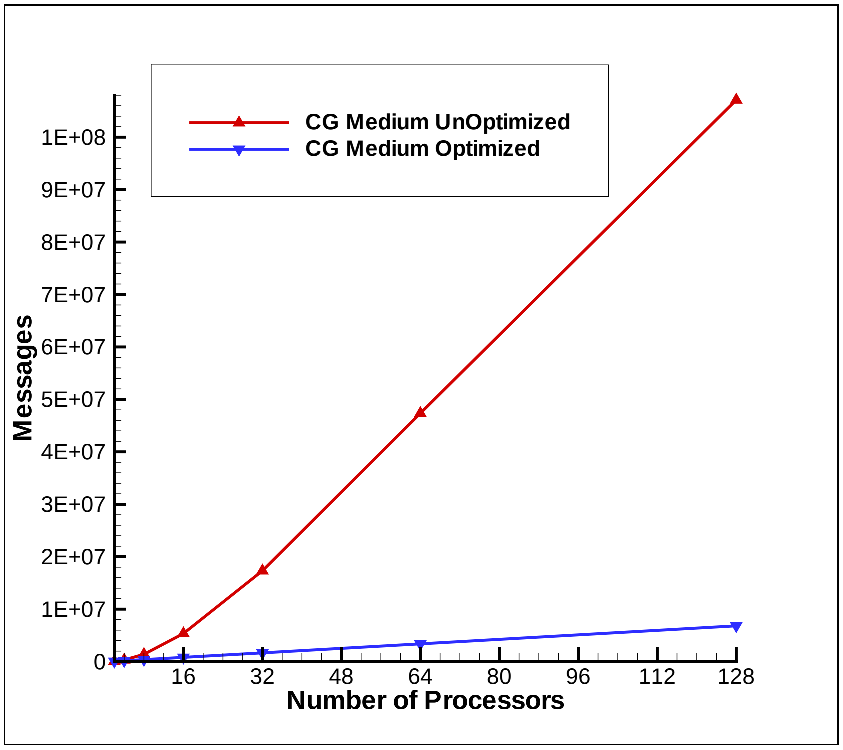

Figure 13. Messages: optimized vs. unoptimized for the medium matrix.

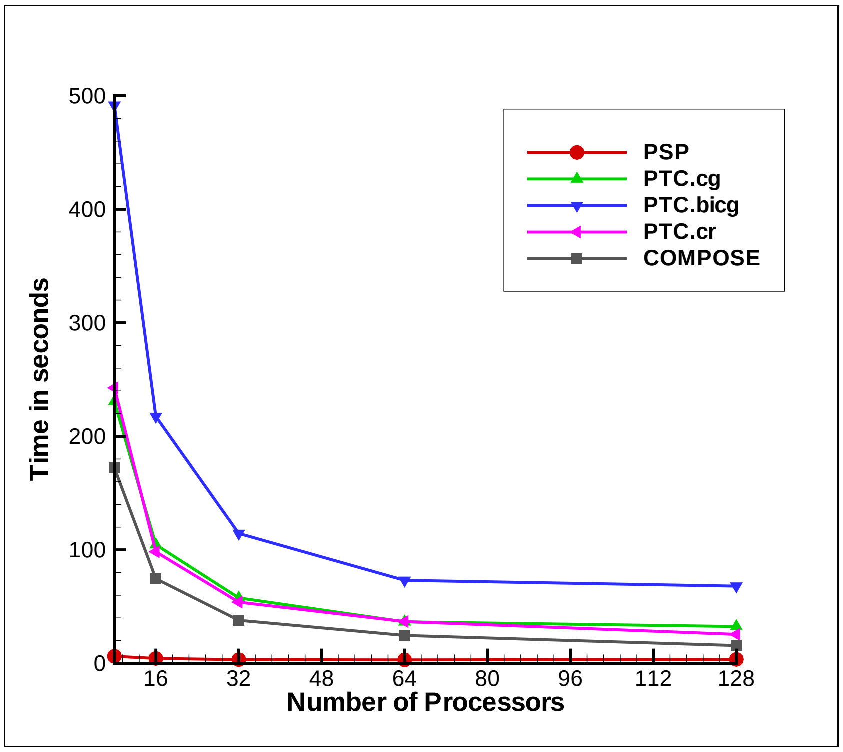

Figure 14. Medium optimized matrix results.

### 4.2.3 Optimized With ATLAS Routines

The same optimized matrix was used for these ATLAS tests. It was assumed that ATLAS could improve the performance of the solvers more when using a larger matrix. When the results came back again, the BCGS method broke down at every processor group except one processor, so the BCGS method results were disregarded. The results were similar to that of the small optimized ATLAS matrix results, in that it helped in some cases but hindered performance in others. The performance numbers are given in Table 6. CG improved at all times except for the time at 4 processors; CR improved at all of the times except those at 2, 4, and 128 processors, BICG improved at all times except at 4, 16, and 32 processors; and for PSPASES the only time that improved was that of the 64 processors time.

Table 6. Effects of using ATLAS routines.

| No. of Processors | Percentage Wall Clock Change (Optimized Matrix) | | |
|---|---|---|---|
| | | PETSc | |
| | PSPASES | CG | CR |
| 4 | −3.8 | −1.4 | −5.6 |
| 8 | −4.1 | +3.1 | +3.7 |
| 16 | −2.2 | +3.1 | −3.1 |
| 32 | −6.6 | +2.3 | −0.6 |
| 64 | +24.9 | +1.2 | +1.1 |
| 128 | −5.8 | +9.1 | +7.4 |

When comparing the methods from 4 processors to 128, CG improved by an overall average of ≈3.8% over the processor times that improved, and its largest improvement came at 128 processors where it improved by ≈9.1%, but at 4 processors the time increased by ≈1.4% (Figure 15). BICG improved by an overall average of ≈4.1% over the processor times that improved, and its largest improvement came at 128 processors where it improved ≈7.4%, but had an overall average increase in time of ≈3.1% over the processor times that did not improve, and the largest increase in time at 4 processors, which was ≈5.6%. CR improved by an overall average of ≈2% over the processor times that improved, with its largest improvement coming at 8 processors where it improved by ≈5%, but had an overall increase in time of ≈8.5%, with the largest increase of ≈14.7% at 128 processors.

PSPASES had an improvement only at 64 processors (≈24.9%). The average increase in the rest of the times was ≈4.5% with the largest increase at 32 processors (≈6.6%). All of the results can be seen in Figure 16. PSPASES outperformed the rest of the solvers with CR and CG following close behind (Figure 17).

### 4.2.4 Unoptimized With ATLAS Routines

For the larger matrix, the decision was made to use ATLAS on the unoptimized matrix to see if it could improve the results that we have received from the regular unoptimized matrix. There was not an improvement at all, even though it did not give any error messages, all of the methods

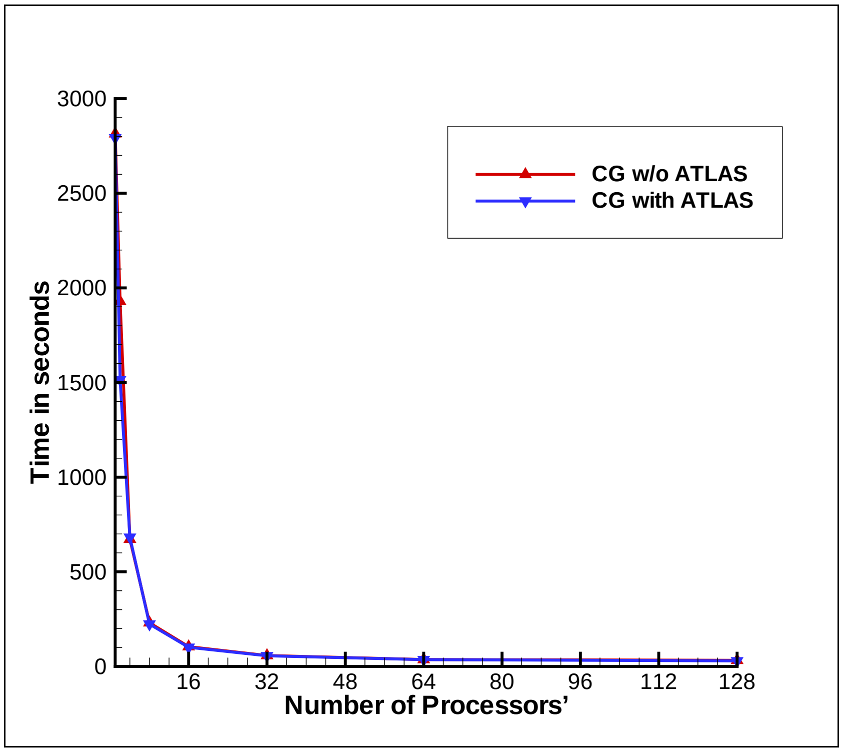

Figure 15.  CG:  ATLAS vs. non-ATLAS for the medium matrix.

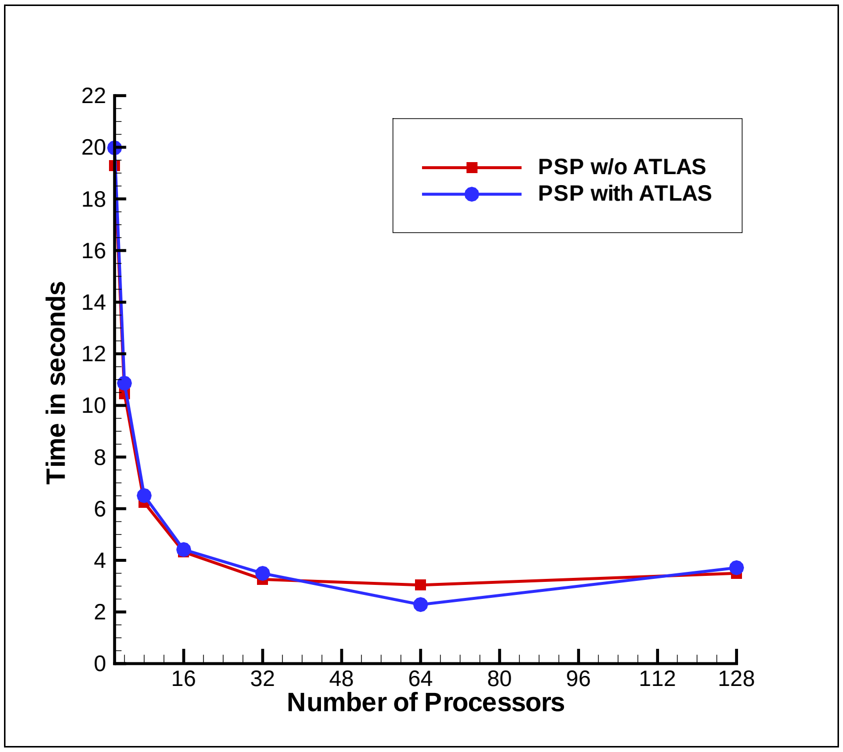

Figure 16.  PSPASES:  ATLAS vs. non-ATLAS for the medium matrix.

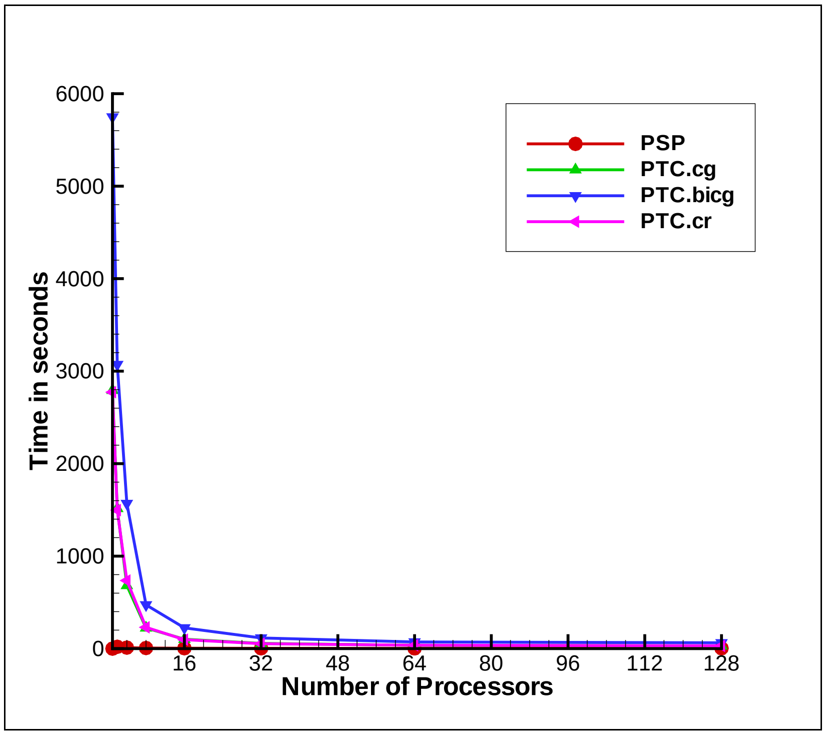

Figure 17. Medium optimized ATLAS matrix results.

diverged from the solution. It was interesting to find that by using this software the results went from valid numbers to invalid numbers. The only package that worked with the unoptimized ATLAS matrix was PSPASES, and the times were not that different from the regular unoptimized matrix results. This software package has helped a little before but for this size and type of matrix, it does not work well with the PETSc methods.

### 4.2.5 Summary

The timing results for this larger problem are shown in Figure 18. Again we were satisfied with the results that we received from testing on the various varieties of the medium matrix, even though we received an unexpected surprise when testing the unoptimized ATLAS matrix. All packages and methods worked well; again VSS outperformed all of the PETSc methods as it was thousands of seconds faster at one processor for both the optimized and unoptimized matrix results. Even at one processor, VSS still was not as fast as PSPASES using two processors. We found again that by using the optimized matrices we save much more time, and we also found that when using a larger matrix the packages scale better. Once again PSPASES outperforms all of the other packages, with COMPOSE coming next, and PETSc's CR and CG following close behind.

The efficiencies of the approaches using this larger matrix are shown in Table 7. These values were calculated using 16 processors as the baseline. Once again, in comparing the relative efficiency between PSPASES and the PETSc methods starting at 16 processors, one can see that

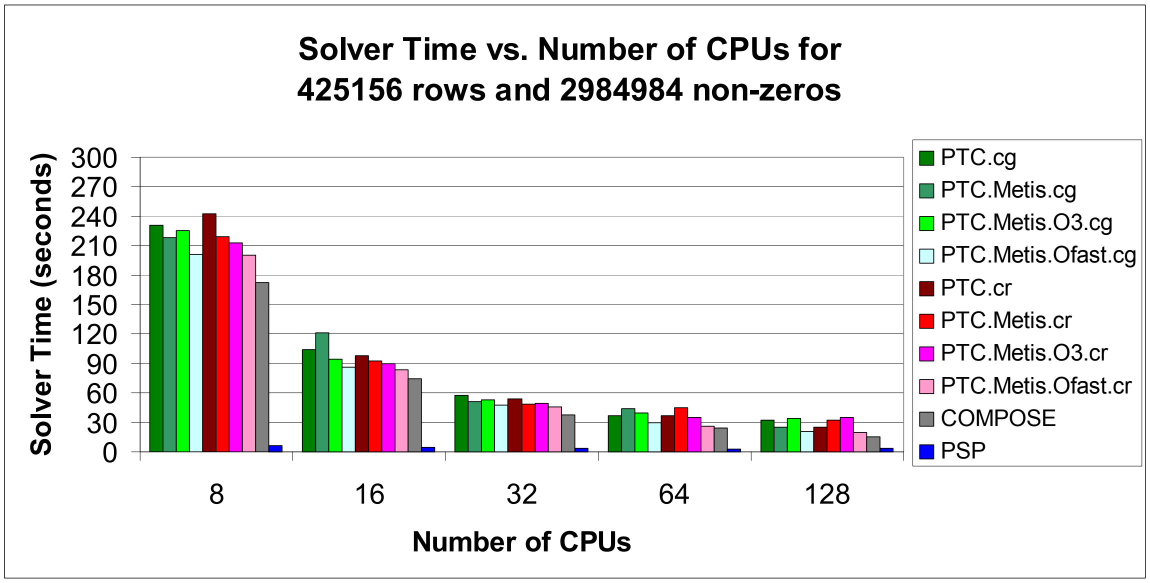

Figure 18. Medium optimized matrix results including those with compiler options.

Table 7. Relative efficiencies of the various solution methods for the medium matrix problem.

| No. of Processors | PSPASES | PETSc | | COMPOSE |
|---|---|---|---|---|
| | | CG | CR | |
| 16 | 1.0 | 1.0 | 1.0 | 1.0 |
| 32 | 0.66 | 0.90 | 0.92 | 0.98 |
| 64 | 0.36 | 0.73 | 0.81 | 0.76 |
| 128 | 0.15 | 0.51 | 0.52 | 0.59 |

PSPASES is far less efficient than the PETSc methods are at 128 processors. This means that at some point CG and CR could potentially catch up to the performance of PSPASES as the CPU sets get larger.

## 5. Conclusion

It is safe to conclude that PSPASES is the fastest solver package out of the packages that we have tested for less than 500,000 nodes and up to 128 processors. Based on these results, the solvers used in the COMPOSE modeling and simulation package are being modularized to allow for a library of solvers from which to choose. The original COMPOSE solver performed well compared to the other packages, even outperforming the PETSc methods. However, PSPASES represents an ideal choice for use in the flow simulation component of COMPOSE. Once the COMPOSE PCG solver is replaced by PSPASES, performance will improve dramatically.

The utility of PSPASES is currently limited to positive, definite, and symmetric matrices. Accordingly, each of the solver packages could be just as valuable as PSPASES depending upon the type of matrix and linear system. New modeling and simulation requirements will require the solution of unsymmetric matrix systems of equations coupled with other matrix systems.

This study has helped us gain insights into the intricacies of linear system solvers (efficiencies, optimal processor pool sizes, error, etc.) that will prove valuable in future research and as we expand current modeling and simulation capabilities.

In the near term, we will be analyzing the performance on two additional architectures. The first is the IBM[*] SP4 system that has a larger set of on-node processors compared to the SGI Origin 3800. The second is an Intel[†] cluster that, by its design, will require a larger processor interconnection network. These results will provide the required knowledge set to allow for automated selection of the optimal approach given the processing configuration.

[*] IBM is a registered trademark of International Business Machines.

[†] Intel is a registered trademark of Intel Corporation.

## 6. References

1. Golub, G.; Ortega, J. M. *Scientific Computing: An Introduction With Parallel Computing*; Academic Press: San Diego, CA, 1993.

2. Zienkiewicz, O. C.; Taylor, R. L. *The Finite Element Method*, 4th ed.; McGraw-Hill Book Company: Wales, UK, 1989; Vol. 1.

3. Kumar, V.; Grama, A.; Gupta, A.; Karypis, G. *Introduction to Parallel Computing*; Benjamin/Cummings Publishing Company Inc.: Redwood City, CA, 1994.

4. Falgout, R.; Jones, J. E. Overview: The Need for Scalable Algorithms. Center for Applied Scientific Computing, 2001; http://www.llnl.gov/CASC/sc2001_fliers/SLS/SLS01.html (accessed June 2003).

5. Balay, S.; Gropp, W.; McInnes, L. C.; Smith, B. *PETSc User's Manual*; Argonne National Laboratory: Argonne, IL, 2001.

6. Joshi, M.; Karypis, G.; Kumar, V.; Gupta, A.; Gustavson, F. *PSPASES: Scalable Parallel Direct Solver Library for Sparse Symmetric Positive Definite Linear Systems User's Manual*; University of Minnesota: Minneapolis, MN, May 1999.

7. Storaasli, O. *Performance of NASA Equation Solvers on Computational Mechanics Applications*; National Aeronautics and Space Administration: Washington, DC, 1996.

8. Henz, B. J.; Shires, D. R.; Mohan, R. V. A Composite Manufacturing Process Simulation Environment (COMPOSE) Utilizing Parallel Processing and Object-Oriented Techniques. *Proceedings of the ICPP-HPSECA*, Vancouver, British Columbia, Canada, August 2002.

9. Shires, D. R.; Mohan, R. V.; Henz, B. J. Scalable Performance and Application of CHSSI Software COMPOSE for Composite Material Processing. *Proceedings of the DOD HPCMO User's Group Conference*, Austin, TX, June 2002.

10. Law, K. H. A Parallel Finite Element Solution Method. *Computers & Structures* **1985**, *23* (6), 845–858.

11. Kanapady, R. Parallel Implementation of Large-Scale Finite Element Computations on a Multiprocessor Machine: Applications to Process Modeling and Manufacturing of Composites. Master's thesis, University of Minnesota, Minneapolis, MN, 1998.

12. Barrett, R.; Berry, M.; Chan, T. F.; Demmel, J.; Donato, M.; Dongarra, J.; Eijkhout, V.; Roldan, P.; Romine, C.; Van der Vorst, H. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*; Society for Industrial and Applied Mathematics: Philadelphia, PA, 1994.

13. Karypis, G.; Kumar, V. METIS: *A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings for Sparse Matrices*. University of Minnesota and the Army HPC Research Center: Minneapolis, MN, 1997.

INTENTIONALLY LEFT BLANK.

| NO. OF COPIES | ORGANIZATION |
|---|---|
| 2 | DEFENSE TECHNICAL INFORMATION CENTER DTIC OCA 8725 JOHN J KINGMAN RD STE 0944 FT BELVOIR VA 22060-6218 |
| 1 | COMMANDING GENERAL US ARMY MATERIEL CMD AMCRDA TF 5001 EISENHOWER AVE ALEXANDRIA VA 22333-0001 |
| 1 | INST FOR ADVNCD TCHNLGY THE UNIV OF TEXAS AT AUSTIN 3925 W BRAKER LN STE 400 AUSTIN TX 78759-5316 |
| 1 | US MILITARY ACADEMY MATH SCI CTR EXCELLENCE MADN MATH THAYER HALL WEST POINT NY 10996-1786 |
| 1 | DIRECTOR US ARMY RESEARCH LAB AMSRL D DR D SMITH 2800 POWDER MILL RD ADELPHI MD 20783-1197 |
| 1 | DIRECTOR US ARMY RESEARCH LAB AMSRL CS IS R 2800 POWDER MILL RD ADELPHI MD 20783-1197 |
| 3 | DIRECTOR US ARMY RESEARCH LAB AMSRL CI OK TL 2800 POWDER MILL RD ADELPHI MD 20783-1197 |
| 3 | DIRECTOR US ARMY RESEARCH LAB AMSRL CS IS T 2800 POWDER MILL RD ADELPHI MD 20783-1197 |

| NO. OF COPIES | ORGANIZATION |
|---|---|
|  | ABERDEEN PROVING GROUND |
| 2 | DIR USARL AMSRL CI LP (BLDG 305) AMSRL CI OK TP (BLDG 4600) |

| NO. OF COPIES | ORGANIZATION |
|---|---|
| 1 | DELAWARE STATE UNIV<br>DR H UMOH<br>ETV BLDG RM 106<br>1200 NORTH DUPONT HWY<br>DOVER DE 19901 |